

Introduction to Machine Learning

placeholder

1 Regression

1.1 Supervised Learning

Supervised Learning is the task of 'learning' a function relationship, based on a given set of inputs/outputs.

Some terminology:

$x \in \mathbb{R}^d$	Inputs (Attributes/Covariates)
$\phi(x) \in \mathbb{R}^p$	Features
$y \in \mathbb{R}$	Outputs (Targets/Labels)
$D = \{(x_i, y_i)\}_{i=1}^n$	Training Set
D'	Test Set
$f: \mathbb{R}^p \rightarrow \mathbb{R}$	Predictor (Model)
$l(f(x), y)$	Loss

Machine Learning Pipelines can often be classified using:

F	Function Class
$L(f)$	Training Loss
	Optimization Method

The function class F is a set of parametrized functions. We are looking for the $f \in F$ that minimizes $L(f)$.

D. Training Loss

$$L(f) := \frac{1}{n} \sum_{i=1}^n l(f(x_i), y)$$

1.2 Multiple Linear Regression

Multiple Linear Regression directly uses the $x \in \mathbb{R}^d$.

Here, $F_{\text{affine}} = \{f(x) = w^\top x + w_0 \mid w \in \mathbb{R}^d, w_0 \in \mathbb{R}\}$.

Rmk. Why are we using linear functions instead?

Any estimator $f \in F_{\text{affine}}$ can be rewritten as $f((x, 1)) = (w, w_0)^\top \cdot (x, 1)$, thus we can augment the inputs $x \mapsto (x, 1)$ and instead search in $F_{\text{linear}} = \{f(x) = \hat{w}^\top x \mid \hat{w} \in \mathbb{R}^{d+1}\}$

Loss Functions

D. Squared Loss $l(f(x), y) := (f(x) - y)^2$

Most common Loss Function, but sensitive to outliers.

D. Absolute Loss $l_{\text{abs}}(f(x), y) := |f(x) - y|$

Less sensitive to outliers, but not differentiable.

D. Huber Loss

$$l_{\text{huber}}(f(x), y) := \begin{cases} \frac{1}{2}(f(x) - y)^2 & |f(x) - y| \leq \delta \\ \delta(|f(x) - y| - \frac{1}{2}\delta) & |f(x) - y| > \delta \end{cases}$$

Using parameter δ , the penalization of outliers can be controlled

Assymetric Loss: In some cases it is desirable to penalize overestimation harder than underestimation, or vice versa.

D. Quantile Loss

$$l_\tau(f(x), y) := \tau \max\{y - f(x), 0\} + (1 - \tau) \max\{f(x) - y, 0\}$$

Using parameter τ , over/underestimation can be penalized

Linear Regression

To find $\hat{f} := \arg \min_{f \in F_{\text{linear}}} L(f)$ we just look for $w \in \mathbb{R}^d$.

$$\hat{w} := \arg \min_{w \in \mathbb{R}^d} L(f_w) = \frac{1}{n} \sum_{i=1}^n \underbrace{(y_i - w^\top x_i)^2}_{l(f(x_i), y_i)}$$

A natural abuse of notation here is $L(w) := L(f_w)$.

This can be rewritten in matrix notation:

$$\sum_{i=1}^n (y_i - w^\top x_i)^2 = \|y - Xw\|^2$$

The factor $\frac{1}{n}$ is irrelevant for Optimization, it doesn't depend on w

So we find the usual problem:

$$\hat{w} = \arg \min_{w \in \mathbb{R}^d} \|y - Xw\|^2$$

The solution is a stationary point, so:

$$\nabla_w \|y - Xw\|^2 = 2X^\top(X\hat{w} - y) = 0$$

Which yields the **Normal Equation** from linear algebra.

$$X^\top X \hat{w} = X^\top y$$

2 Classification

In regression, we search an $\hat{f} : \mathbb{R}^d \rightarrow \mathbb{R}$, i.e. $y, \hat{y} \in \mathbb{R}$.
In classification, we want $\hat{y} \in \mathcal{Y} \subset \mathbb{R}$, s.t. \mathcal{Y} is discrete.

2.1 Binary Classification

We generally use $\mathcal{Y} = \{+1, -1\}$ and set $\hat{y} = \text{sgn}(\hat{f}(x))$.
So, a linear classifier where $\hat{f}(x) = w^\top x$ takes the form:

$$x \mapsto \begin{cases} 1 & w^\top x > 0 \\ -1 & w^\top x < 0 \end{cases}$$

D. Decision Boundary $\{x \in \mathbb{R}^d \mid \hat{f}(x) = 0\}$

Like in regression, using features is again possible.

2.2 Surrogate Loss

We'd like to reuse the loss minimization from regression.
A natural metric for accuracy is simply checking if $\hat{y} = y$.

D. Zero-One Loss

$$l_{0-1}(\hat{y}, y) := \mathbb{I}_{\hat{y} \neq y} = \begin{cases} 1 & \hat{y} \neq y \\ 0 & \hat{y} = y \end{cases}$$

We could try minimizing this:

$$\sum_{(x,y) \in \mathcal{D}} l_{0-1}(\hat{y}, y) = \sum_{(x,y) \in \mathcal{D}} \mathbb{I}_{f_w(x) \cdot y < 0}$$

Unfortunately, l_{0-1} is non-continuous and non-convex.
We introduce *surrogate loss* to still apply GD.

Note how $\mathbb{I}_{\hat{y} \neq y} = \mathbb{I}_{\hat{y} \cdot y < 0}$, so l_{0-1} only depends on $z := \hat{y} \cdot y$.
We thus define losses over z , that are cont. and convex.

D. Surrogate Loss

$$l_{\text{exp}} = e^{-z} \quad l_{\log} = \log(1 + e^{-z})$$

A notable difference is that l'_{exp} is unbounded,
while $l'_{\log} = \frac{1}{1+e^z} \in (-\frac{1}{2}, -1)$ for $z < 0$.
This is better for outliers, thus l_{\log} is usually preferred.

2.3 Logistic Regression

We assume $w_0 = 0$

We try to minimize $l_{\log} = \log(1 + e^{-z})$, so:

$$L(w) = \frac{1}{n} \sum_{i=1}^n l_{\log}(z_i) = \frac{1}{n} \sum_{i=1}^n \log\left(1 + e^{-\overbrace{y_i \cdot w^\top x_i}^{z_i}}\right)$$

Assume $\{x_i, y_i\}_{i=1}^n$ is linearly separable, i.e.

$$\exists w \in \mathbb{R}^d : \underbrace{y_i \cdot w^\top x_i}_{z_i} > 0 \quad \forall i \leq n$$

Then there are multiple valid decision boundaries. Additionally, $L(w)$ is then convex.

the distance x_0 to the decision boundary is: $\|x_0\|_2 \cdot |\cos(\theta)|$.
 θ between $w, x_0 \in \mathbb{R}^d$

$$\|x_0\|_2 \cdot |\cos(\theta)| = \|x_0\|_2 \cdot \frac{|w^\top x_0|}{\|w\|_2 \cdot \|x_0\|_2} = \frac{|w^\top x_0|}{\|w\|_2}$$

Note if w is a unit-vector, this is just $|w^\top x_0|$

D. Margin $\text{margin}(w) := \min_{1 \leq i \leq n} y_i \cdot w^\top x_i$

2.4 Solutions

D. Maximum Margin Solution

$$w_{\text{MM}} := \max_{\|w\|_2=1} \min_{1 \leq i \leq n} (y_i \cdot w^\top x_i)$$

If \mathcal{D} is linearly separable, this is convex.

Rmk. Also called *hard-margin SVM*, assuming lin.-sep. data

D. Support Vector Machine

$$w_{\text{SVM}} := \min_{w \in \mathbb{R}^d} \|w\|_2 \quad \text{s.t.} \quad y_i \cdot w^\top x_i \geq 1 \quad \forall i \leq n$$

Solving these problems is actually equivalent, up to scaling:

L. $\frac{w_{\text{SVM}}}{\|w_{\text{SVM}}\|_2} = w_{\text{MM}}$ (This also holds for the case $w_0 \neq 0$)

By relaxing the SVM constraints, we can use the SVM problem on lin.-insep. data too:

$$y_i \cdot w^\top x_i \leq 1 \quad \rightarrow \quad y_i \cdot w^\top x_i \leq 1 - \zeta$$

$\zeta = (\zeta_1, \dots, \zeta_n)$ s.t. $\zeta_i \geq 0$

D. Soft-margin SVM

$$w_{\text{SM}} = \min_{w \in \mathbb{R}^d, \zeta \in \mathbb{R}^n} \left(\|w\|^2 + \lambda \sum_{i=1}^n \zeta_i \right) \quad \text{s.t.} \quad \begin{cases} y_i \cdot w^\top x_i \geq 1 - \zeta_i \\ \zeta_i \geq 0 \quad \forall i \leq n \end{cases}$$

λ (hyperparam.) intuitively controls how much a violation is penalized

Another perspective: The optimal ζ_i are:

$$\zeta_i = \begin{cases} 1 - y_i \cdot w^\top x_i & \text{if } y_i \cdot w^\top x_i \leq 1 \\ 0 & \text{else} \end{cases}$$

So the problem can be formulated without ζ too:

D. l_2 -penalized Hinge Loss Optimization

$$\min_{w \in \mathbb{R}^d} \left(\|w\|^2 + \lambda \underbrace{\sum_{i=1}^n \max(0, 1 - y_i \cdot w^\top x_i)}_{\text{Hinge Loss}} \right)$$

2.5 Gradient Descent for Classification

In practice, instead of explicitly solving w_{SVM} or w_{MM} , GD is usually applied on a diff.-able convex surrogate loss.

2.5.1 On linearly seperable data

Assuming $\{x_i, y_i\}_{i=1}^n$ is lin. seperable, $L(w) = \frac{1}{n} \sum_{i=1}^n l_{\log}(z_i)$ is convex, but no global optimum exists. Using GD, $L(w)$ will approach 0, but the iterates $\{w^t \mid t \in \mathbb{N}\}$ diverge. However, w^t may converge *in direction*, and interestingly:

Th. GD converges to w_{MM} for lin.-sep. data (log. loss)

$$\lim_{t \rightarrow \infty} \frac{w^t}{\|w^t\|} = w_{\text{MM}}$$

On $L(w)$ (logistic regression), $\mu = 1$

2.5.2 On linearly inseperable data

Assuming $\{x_i, y_i\}_{i=1}^n$ is strictly lin. inseperable, i.e.

$$\forall w \neq 0 : \exists i \leq n : y_i \cdot w^\top x_i < 0$$

Th. GD converges on lin.-insep. data (log. loss)

$$\exists \hat{w} \in \mathbb{R}^d : \lim_{t \rightarrow \infty} w^t = \hat{w}$$

On $L(w)$ (logistic regression), $\mu = \frac{4}{\sigma_{\max}^2(X)}$

Rmk. Only holds for l_{\log} . In general, this is a hard problem.

2.6 Multiclass Classification

What if $|\mathcal{Y}| > 2$? E.g. $\mathcal{Y} = \{\text{cat}, \text{dog}, \text{fish}\}$

Idea: Train $K := |\mathcal{Y}|$ classifiers $\hat{f}_1, \dots, \hat{f}_K \in F$.

Why not one \hat{f} ? E.g. discretize further: $1 \mapsto \text{cat}, 2 \mapsto \text{dog}, 3 \mapsto \text{fish}$.

Problem: this assignment suggests cats are closer to dogs than fish.

We can then predict the class using these \hat{f}_k :

$$\hat{y}(x) = \arg \max_{1 \leq k \leq K} \hat{f}_k(x)$$

This leads to one decision boundary per class.

D. One-vs-Rest Training

Train each model seperately by relabeling for each \hat{f}_k :

1. Define $\mathcal{D}_k = \{x_i, \tilde{y}_i\}$ where $\tilde{y}_i := \begin{cases} -1 & y_i = k \\ 1 & y_i \neq k \end{cases}$
2. Run binary classification on \mathcal{D}_k to get \hat{f}_k

This leads to K classification problems, which might be slow

Another way to reuse the existing methodology is to use a new loss:

D. Cross-Entropy Loss

$$l_{\text{ce}}(\hat{f}_1(x), \dots, \hat{f}_K(x), y) = -\log\left(\frac{\exp(\hat{f}_y(x))}{\sum_{k=1}^K \exp(\hat{f}_k(x))}\right)$$

$$y \in \{1, \dots, K\}, \quad \hat{f}_i(x) \in \mathbb{R}$$

l_{ce} encourages the *true class* $\hat{f}_{y_i}(x_i)$ to be the largest $\hat{f}_k(x_i)$.

Rmk. For $K = 2$, if we use $\mathcal{Y} = \{+1, -1\}$ then $l_{\text{ce}} = l_{\log}$.

The parametrized optimization problem then is:

$$\min_{w_1, \dots, w_K \in \mathbb{R}^d} \left(\sum_{i=1}^n l_{\text{ce}}(f_w(x_i), y_i) \right)$$

Here, $w \in \mathbb{R}^{d \times K}$ is a matrix: $w = (w_1, \dots, w_K)$

This then yields $\hat{f}_k = f_{\hat{w}_k}$

Rmk. These methods may lead to very different decision boundaries!

2.7 Generalization

First, a lot of assumptions:

1. Inputs $X \in \mathcal{X}$ come from a prob. distribution \mathbb{P}_X
2. Training & Test set sampled i.i.d. from same distribution
Note that in general, this is rarely true.
3. There exists a ground-truth y^*
4. The observed labels are noisy: $(y \mid x) = \epsilon \cdot y^*(x)$
A *multiplicative* noise model, unlike lin.-reg. : $\mathcal{Y} = f^*(X) + \epsilon$
5. ϵ is also from a prob. distribution \mathbb{P}_ϵ
Not necessarily indep. from x !

Focusing on $y^*(x) \in \{+1, -1\}$, we set $\epsilon \in \{+1, -1\}$.

Intuitively: ϵ just flips the label

This allows defining a Joint-Distribution

$$\mathbb{P}[x, y] = \mathbb{P}_x[x] \cdot \mathbb{P}[y \mid x]$$

2.7.1 Evaluation

An intuitive metric to check is proximity to y^* :

Which can be done using the 0-1-loss

$$l(\hat{y}(x), y^*(x)) = \mathbb{I}_{\hat{y}(x) \neq y^*(x)}$$

Now, we can define the expected classification error:

$$\mathbb{E}_X[l(\hat{y}(x), y^*(x))] = \mathbb{E}_X[\mathbb{I}_{\hat{y}(x) \neq y^*(x)}] = \mathbb{P}[\hat{y}(x) \neq y^*(x)]$$

We can't compute or estimate this, since we don't have y^* . However, we can find an estimate of $\mathbb{E}_{X,Y}[l(\hat{y}(X), Y)]$ using the observed X, Y .

Why is this useful? It approximates the generalisation error:

D. Generalisation Error (0-1-Loss)

$$L(\hat{f}; \mathbb{P}_{X,Y}) = \mathbb{E}_{X,Y}[l(\hat{f}(X), Y)] = \mathbb{P}_{X,Y}(\hat{y} \neq y)$$

We can empirically evaluate this on a test set:

$$\frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(x,y) \in \mathcal{D}_{\text{test}}} \mathbb{I}_{\hat{y}(x) \neq y^*(x)}$$

2.8 Hypothesis Testing

Classical statistical methods can also be used.

D. Asymmetric Errors

Misclassifications may be weighted differently.

Mistakenly classifying x with $y^*(x) = 1$ as 2, may be worse than 3.

For binary classification, we may label:

$$+1 \mapsto \text{"Positive"} \quad -1 \mapsto \text{"Negative"}$$

This leads to the notion of confusion matrices.

	$y = -1$	$y = +1$
$\hat{y} = -1$	TN	FN (Type II)
$\hat{y} = +1$	FP (Type I)	TP

D. Empirical Measure

For an event $A \subset \mathcal{X} \times \{+1, -1\}$

$$\mathbb{P}_n[A] := \frac{1}{n} \sum_{i=1}^n \mathbb{I}_{(x_i, y_i) \in A}$$

$$\mathcal{D} = \{(x_i, y_i) \mid i \leq n\} \subset \mathcal{X} \times \{+1, -1\}$$

$\mathbb{P}_n[A]$ is the percentage of $(x, y) \in \mathcal{D}_{\text{train}}$ that belong to A .

L. $\mathbb{P}_n[A]$ is an estimate of $\mathbb{P}_{X,Y}[A]$:

$$\lim_{n \rightarrow \infty} \mathbb{P}_n[A] = \mathbb{P}_{X,Y}[A] \quad (\text{Law of large numbers})$$

Rmk. Asymmetric Loss in binary classification

We can now weigh FP, FN differently in the 0-1-error:

$$\frac{c_{FN}}{|\{x \mid y = +1\}|} \underbrace{\sum_{(x,y), y=1} \mathbb{I}_{\hat{y} \neq +1}}_{\#FN} + \frac{c_{FP}}{|\{x \mid y = -1\}|} \underbrace{\sum_{(x,y), y=-1} \mathbb{I}_{\hat{y} \neq -1}}_{\#FP}$$

Here c_{FP}, c_{FN} are the weights for penalization

Generally, reducing FP errors increases FN errors, and vice versa.

2.9 ROC Curves

Rmk. A side-effect of using $\hat{y}(x) = \text{sign} \hat{f}(x)$ is that the magnitude $|\hat{f}(x)|$ can be interpreted as *confidence*.

We can set:

$$\hat{y}_\tau(x) = \text{sign}(\hat{f}(x) - \tau) = \begin{cases} +1 & \text{if } \hat{f}(x) > \tau \\ -1 & \text{if } \hat{f}(x) < \tau \end{cases}$$

Now τ can be used to penalize FP ($\tau > 0$) or FN ($\tau < 0$).

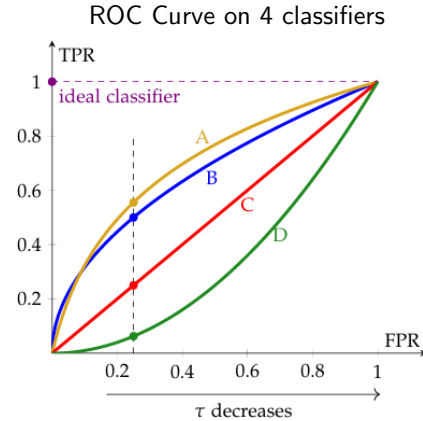
Note how this way, we don't modify the Optimization problem.

What if we don't know which FP/TP rate is desired?

Formally: which τ should be used?

D. ROC Curve (Receiver Operating Characteristic)

Plots TP Rate against FP Rate for different τ .



Introduction to Machine Learning (2026), p. 160

Rmk. **How to read this?** A straight line is equivalent to random guessing, anything above is better. τ isn't directly included in the curve, but it follows from the definition that τ decreases as the FP rate increases.

How can we measure performance independent of τ ?

D. AUROC (Area under ROC)

AUROC is 1 for the ideal classifier, and always in $[0, 1]$.

3 Kernels

Motivation: Regression using feature maps $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^p$:

$$\min_{w \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^n l(y_i, w^\top \cdot \phi(x_i))$$

What if computing/storing $\phi(x)$ is expensive/infeasible?

e.g. if p is large, or infinite

Rmk. To store a poly. $p(x) : \mathbb{R}^d \rightarrow \mathbb{R}$ with $\deg(p) = m$ we require $p = \mathcal{O}(d^m)$ features. Storing n data points requires $\mathcal{O}(nd^m)$ memory. Not good.

3.1 Kernelization

By constraining w to $\text{span}(\Phi^\top) \subset \mathbb{R}^p$ we can drastically improve memory usage. Since we know a minimizer exists here, we don't "lose anything".

D. Kernelization

1. **Reparametrization:** We assume $w = \Phi^\top \alpha$ (i)

2. **Loss via Inner Products:** Observe:

$$f(x) = w^\top \phi(x) \stackrel{(i)}{=} (\Phi^\top \alpha)^\top \phi(x) = \sum_{i=1}^n \alpha_i (\phi(x_i)^\top \phi(x))$$

Note: x_i only appears in *inner products* $\phi(x_i)^\top \phi(x_j)$

3. **Replace Inner Products:** We define:

$$k : \begin{cases} \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R} \\ k(x, x') = \phi(x)^\top \phi(x') \end{cases} \quad K : \begin{cases} K \in \mathbb{R}^{n \times n} \\ K_{ij} = k(x_i, x_j) \end{cases}$$

Now, we can reformulate the optimization problem:

$$\min_{\alpha \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^n l\left(y_i, \sum_{j=1}^n \alpha_j k(x_i, x_j)\right) = \min_{\alpha \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^n l(y_i, (K\alpha)_i)$$

By storing $K \in \mathbb{R}^{n \times n}$ instead of $\phi(x) \in \mathbb{R}^p$ for $i = 1, \dots, n$, the memory usage is reduced: $\mathcal{O}(np) \rightarrow \mathcal{O}(n^2)$.

3.2 The Kernel Trick

Using k , the computation time is still $\mathcal{O}(n^2p)$ if

$$k(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$$

So let's replace k with a simple function, which guarantees the existence of some ϕ (which we never calculate).

Rmk. Since we only *implicitly* specify ϕ via k , we can use ϕ s.t. $p = \infty$ now.

D. Kernel Function $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$

1. k is symmetric: $\forall x, x' : k(x, x') = k(x', x)$
2. k is PSD: $\forall n \in \mathbb{N}, \forall (x_1, \dots, x_n) \in \mathbb{R}^d$:

$$K = \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{bmatrix} \text{ is PSD}$$

Th. Kernels guarantee existence of ϕ

If k is a kernel, there exists a Hilbert Space $(\mathcal{H}, \langle \cdot, \cdot \rangle_{\mathcal{H}})$ s.t.

$$\exists \phi : \mathbb{R}^d \rightarrow \mathcal{H} \text{ s.t. } k(x, x') = \left\langle \phi(x), \phi(x') \right\rangle_{\mathcal{H}} \quad \forall x, x' \in \mathbb{R}^d$$

\mathcal{H} may be, for example, \mathbb{R}^p with $\|\cdot\|_2$.

L. Properties of Kernels

1. Composed feature maps are Kernels

$$\left. \begin{array}{l} \phi : \mathbb{R}^d \rightarrow \mathbb{R}^p \\ \psi : \mathbb{R}^d \rightarrow \mathbb{R}^p \end{array} \right\} \quad k(x, x') = \left\langle \psi(\phi(x)), \psi(\phi(x')) \right\rangle$$

2. Kernels can be added in 2 ways, yielding a kernel

$$\begin{aligned} \text{(i)} \quad k((x, y), (x', y')) &= k_1(x, x') + k_2(y, y') \\ \text{(ii)} \quad k(x, x') &= k_1(x, x') + k_2(x, x') \end{aligned}$$

3. Kernels can be multiplied in 2 ways, yielding a kernel